Assignment 1 Report

Group 3: Rupesh Acharya, Niyati Maheshwari, Peter Vayda
October 12, 2018

Problem 1

The primary mission of this effort was to automate the process of extracting quarterly company data from the US Securities and Exchange Commission (SEC) Electronic Data Gathering, Analysis, and Retrieval (EDGAR) website and delivering the data in a usable form for the customer. Publically traded companies must file 10-Q forms each quarter (except the 4th quarter when they file the annual 10-K form) to disclose their financial position for public record. Within this filing, there are many tables containing unaudited financial data. As the 10-Q is a large document, the challenge of this effort was determining the tables of interest for the customer and modifying formatting to enable meaningful analysis.

To do this we divided our effort into tasks:

- Generate the 10-Q filing index URL using the central index key (CIK) for a given company and the SEC accession number (specific to each filing)
- Parse the 10-Q filing index page to get the link for the 10-Q document
- Extract tables of interest
- Package tables of interest (make directory, save tables of interest as comma separated values, and compress files as a zip file)
- Upload to Amazon Web Services (AWS) Simple Storage Service (S3) for ease of accessibility by customer

Overall our effort was a success and we were able to create a product to accomplish the above tasks for the customer.

Key Code Functionality

The following code excerpts show some of the key functionality that enables our code to perform the previously mentioned tasks. Note, these are only snippets of code. Please see our GitHub link for our full application code.

Our application starts by having the user input the CIK, SEC Accession number, AWS Key ID, and AWS Secret Key. There are checks to ensure the AWS keys are valid and there is a website associated with the given CIK and SEC Accession nmber. The application then generates the link to the 10-Q index using the CIK and SEC Accession number.

```
x = input("Enter CIK ")
cik = x.lstrip("0")

accNumber = input("Enter Accession Number ")

#pass credentials
accesskey = input("input AWS access key")
secretaccesskey = input("input AWS secret access key")

#YOUR_ACCESS_KEY
aws_access_key_id = accesskey
#YOUR_SECRET_KEY
aws_secret_access_key = secretaccesskey

try:
    s3_connection = boto.connect_s3(aws_access_key_id, aws_secret_access_key)
    conn_check = s3_connection.get_all_buckets()

except:
    print("AWS keys invalid. Please try again")
    sys.exit()


if len(cik) == 0 or len(accNumber) == 0:
    cik = "0000051143"
    accNumber = "0000051143-13-000007"
```

Utilizing the 10-Q index link, we parse the page the page using Beautiful Soup to find the 10-Q document link.

```
logging.debug("Trying to hit the generated URL")
request = requests.get(url)
if request.status_code == 200:
    logging.debug("Status = 200: Now creating link for 10-Q file")
    soup = BeautifulSoup(request.content, "lxml")
    table = soup.find_all('table')[0]
    link = table.find('a')
    url_final = "https://www.sec.gov" + link.get('href')
    print(url_final)
```

With the 10-Q document link, we now parse this page (again using Beautiful Soup) and find all tables tags in the document formatting. With all the tables, we generate a loop to go through all the tables, check the formatting style to identify the tables of interest, and copy content from those tables to an array. Additionally, there is functionality to eliminate extra formatting characters within the text leaving only the English words and numerical figures.

```
    response = requests.get(url_final)
    soup1 = BeautifulSoup(response.text, 'html.parser')
    logging.debug("Finding all tables")
    all_tables = soup1.find_all('table')
    count = 0
    all_header = []
    for t in all_tables:
        logging.debug("Iterating through all tables")
        newTable = []
        logging.debug("Finding all rows in each table")
        trs = t.find_all('tr')
        for tr in trs:
            logging.debug("Checking style of row if it has color")
            if "background" in str(tr.get('style')) or "bgcolor" in str(tr.get('style')) or "background-color" in str
                logging.debug("Background Style found: Finding parent table of the row")
                tr1 = tr.find_parent('table')
                gettable = []
                logging.debug("Iterating through all rows of the parent table")
                gettrs = tr1.find_all('tr')
                for x in gettrs:
                    row = []
                    column = []
                    gettd = x.find_all('td')
                    logging.debug("Iterating for columns of that row")
                    for y in gettd:
                        logging.debug("Fin  ")
                        r = y.text;
                        r = re.sub(r"['()]", "", str(r))
                        r = re.sub(r"[$]", " ", str(r))
                        if len(r) > 1:
                            r = re.sub(r"[—]", "", str(r))
                            column.append(r)
                    row = ([y.encode('utf-8') for y in column])
                    gettable.append(y.decode('utf-8').strip() for y in row)
                newTable = gettable
                break

            else:

                logging.debug("Nothing found row checking styling in Columns")
                tds = tr.find_all('td')
                for td in tds:

                    if "background" in str(td.get('style')) or "bgcolor" in str(tr.get('style')) or "background-colo
                        logging.debug("Background Style found: Finding parent table of the column")
                        td1 = td.find_parent('table')
                        gettable = []
                        gettrs = td1.find_all('tr')
                        for x in gettrs:
                            row = []
                            column = []
                            gettd = x.find_all('td')
```

With the tables of interest identified, the application now formats the array to remove blank space within the array rows. This enhances readability of the tables and enables analysis of numbers by column (without removing blank space some values are shifted off of the column location where they were originally in the 10-Q document).

	As of December 31, 2014	As of September 30, 2015 (unaudited)
Short-Term Portion of Long-Term Debt		
2.125% Notes due on May 19, 2016[1]	$ 0	$ 999
Capital Lease Obligation	10	238
Total	$ 10	$ 1,237
Long-Term Debt		
2.125% Notes due on May 19, 2016	$ 1,000	$ 0
3.625% Notes due on May 19, 2021	1,000	1,000
3.375% Notes due on February 25, 2024	1,000	1,000
Unamortized discount for the Notes above	(8)	(6)
Subtotal	2,992	1,994
Capital Lease Obligation	236	0
Total	$ 3,228	$ 1,994

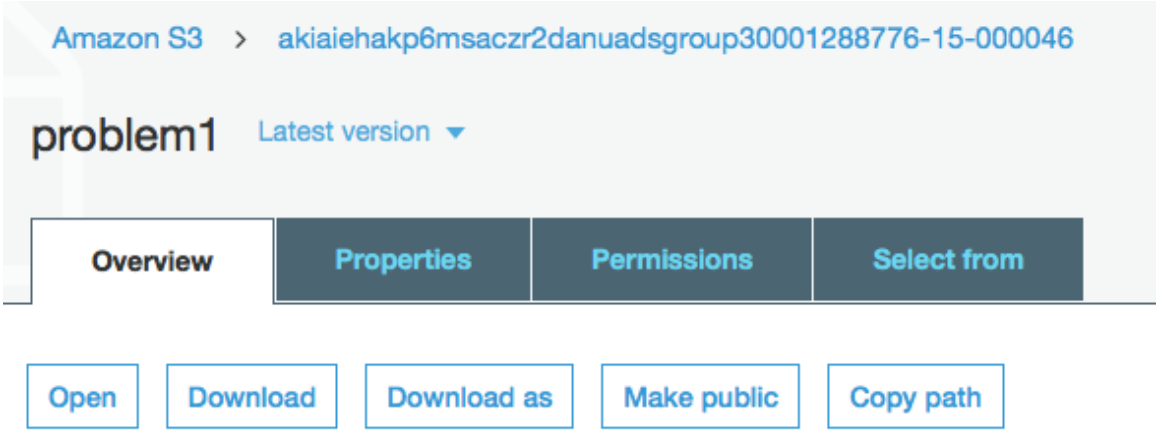
As of Decem	As ofSeptember 30, 2015	
unaudited		
Short-Term Portion of Long-Term Debt		
2.125% Note	999	
Capital Lease	10	238
Total	10	1,237
Long-Term Debt		
2.125% Note	1,000	
3.625% Note	1,000	1,000
3.375% Note	1,000	1,000
Unamortized discount for the Notes above		
Subtotal	2,992	1,994
Capital Lease	236	
Total	3,228	1,994

```
        if not len(newTable) == 0:
            count += 1
            ptag = t.find_previous('p')
            while ptag is not None and checktag(ptag.get('style')) == "false" and len(ptag.text) <= 1:
                logging.debug("Checking Paragraph tags styling for cleansing")
                ptag = ptag.find_previous('p')
                if checktag(ptag.get('style')) == "true" and len(ptag.text) >= 2:
                    global name
                    name = re.sub(r"[^A-Za-z0-9]+", "", ptag.text)
                    if name in all_header:
                        hrcount += 1
                        hrname = name + "_" + str(hrcount)
                        all_header.append(hrname)
                    else:
                        hrname = name
                        all_header.append(hrname)
                        break

            folder_name = createfolder(soup1)
            path = str(os.getcwd()) + "/" + folder_name
            if not os.path.exists(path):
                os.makedirs(path)
            if len(all_header) == 0:
                filename = folder_name + "-" + str(count)
            else:
                filename = all_header.pop()
            csv_name = filename + ".csv"
            csv_path = path + "/" + csv_name
            final_path = path;
            with open(csv_path, 'w', encoding='utf-8-sig', newline='') as f:
                logging.debug("Writing tables to CSV")
                writer = csv.writer(f)
                writer.writerows(newTable)

    logging.debug("Creating Zip file")
    shutil.make_archive("out", 'zip', final_path)
else:
    logging.debug("Error 404: CIK or Accession Number Does not exist!")
    print('Web site does not exist')
```

With the financial data copied and formatted, we create a directory and individual csv files for each table in the document. The directory is then compressed as a zip file for portability.

```
bucketname = accesskey.lower() + "nuadsgroup3"+ accNumber

bucket = s3_connection.create_bucket(bucketname)
logging.debug("Creating AWS S3 bucket " + bucketname)

upload_to = Key(bucket)
upload_to.key = 'problem1'

logging.debug("Zip File & Log File Uploaded to AWS S3 bucket " + bucketname)
upload_to.set_contents_from_filename(str("out" + ".zip"))
upload_to.set_contents_from_filename(str(logName))
```

Finally, we create a unique AWS S3 bucket name based on the AWS Access Key ID and accession umber, create an object called problem 1 on the bucket, and store our zip file and log file within the object. The files are now available for customer use on account where the access key originated.



Problem 2

The primary mission of this effort was to automate the process of extracting individual yearly usage statistics of the US Securities and Exchange Commission (SEC) Electronic Data Gathering, Analysis, and Retrieval (EDGAR) website, performing exploratory data analysis, and delivering the data in a digestible form for the customer. The data is logged daily on the EDGAR usage statistics, which makes for a very large dataset if using 365 days for yearly analysis. To create a subset of the data, our application utilized data from the first day of every month to create the yearly usage data for analysis. Then summary metrics and visualizations were developed to aid in understanding the data.

To do this we divided our effort into tasks:

- Automate access to data of interest
- Cleanse data (replace null values and outliers)
- Compute summary metrics of data categories
- Generate visualizations of data
- Package and upload to Amazon Web Services (AWS) Simple Storage Service (S3) for ease of accessibility by customer

Overall our effort was a success and we were able to create a product to accomplish the above tasks for the customer.

Key Code Functionality

The following code excerpts show some of the key functionality that enables our code to perform the previously mentioned tasks. Note, these are only snippets of code. Please see our GitHub link for our full application code.

Our application starts with our user inputting a year of interest, their AWS Key ID, and associated AWS Secret Key. Based on the year of interest, the URLs of 12 data files (one per month) are generated. Typically the application uses the first day of the month. That being said our application checks the amount of data in the file on the first day and if there isn't data (i.e. corrupt data or national observed holiday), it moves on to the next day in the month until data is found.

```
Quaters = {'Qtr1': ['01', '02', '03'], 'Qtr2': ['04', '05', '06'], 'Qtr3': ['07', '08', '09'], 'Qtr4': ['10', '11', '
days= range(1, 32)
for key,value in Quaters.items():
    for val in value:
        for d in days:
            url= 'http://www.sec.gov/dera/data/Public-EDGAR-log-file-data/' + str(x) + '/'+ str(key)+ '/log'+ str(x)+
            print(url)
            urllib.request.urlretrieve(url, x+'_zip/'+url[-15:])
            logging.info("Retrieving zipped log file")
            if os.path.getsize( x+'_zip/'+url[-15:]) <= 4515:
                os.remove( x+'_zip/'+url[-15:])
                logging.info("Log file is not present for "+ str(d) + " day")
                continue
            break
```

Once the 12 files are downloaded to a local directory, the data is loaded into a pandas dataframe and cleansed for null values. Categorical values like norefer and noagent were turned to true.

```
for file_ in allFiles:
    data = pd.read_csv(file_)
    logging.info("Calculating missing values in coloumns")
    print(data.isnull().sum())
    logging.info("Finding the NaN values in each coloumn")
    logging.info("Exploring Browser Coloumn")
    a = ['mie','fox','saf','chr','sea', 'opr','oth','win','mac','lin','iph','ipd','and','rim','iem']
    [len(list(group)) for key, group in groupby(a)]
    logging.info("Grouping the values of all the browsers and storing in a dataframe")
    df = pd.DataFrame(data,columns = ['browser'])
    d = df.apply(pd.value_counts)
    logging.info("Counting the frequency of each browser type used in descending order")
    list(d.index)
    data['browser'].replace(np.nan,d.index[0], inplace = True) #Replacing the NaN values in the browser colomun with t
    data.isnull().sum()
    logging.info("confirming that no NaN values are present on the browser coloumn")
    logging.info("Working on the Size Coloumn")
    logging.info("Replacing the file size for ext : txt, by the mean of all the file size corresponding to txt")
    s = data[['extention','size']].groupby(data['extention'].str.contains('txt'))['size'].mean().reset_index(name='mea
    data.loc[(data['size'].isnull()) & (data['extention'].str.contains('txt'))] = s
    data.reset_index(drop = True)
    logging.info("Replacing the file size with NaN values for ext : htm, by the mean of all the file size correspondi
    g = data[['extention','size']].groupby(data['extention'].str.contains('htm'))['size'].mean().reset_index(name='mea
    data.loc[(data['size'].isnull()) & (data['extention'].str.contains('htm'))] = g
    data.reset_index(drop=True)
    logging.info("Replacing the file size with NaN values for ext : xml, by the mean of all the file size correspondi
    h = data[['extention','size']].groupby(data['extention'].str.contains('xml'))['size'].mean().reset_index(name='mea
    data.loc[(data['size'].isnull()) & (data['extention'].str.contains('xml'))] = h
    data.reset_index(drop=True)
```

```
    logging.info("To check how many NaN values are remaining ")
    logging.info("Replacing the file size for rest of the files with the mean of file size of txt extension, as it is
    data.loc[data['size'].isnull()] = s
    print(data.isnull().sum())
    logging.info("Working on all other coloumns")
    logging.info("If cik,Accession,ip,date are empty fields drop the records")
    data.dropna(subset=['cik'],inplace=True)
    data.dropna(subset=['accession'],inplace=True)
    data.dropna(subset=['ip'],inplace=True)
    data.dropna(subset=['date'],inplace=True)
    data.dropna(subset=['time'],inplace=True)
    logging.info("Calculating the max categorical value in other coloumns( code, zone,extention,idx,find) and filling
    data['code'].fillna(data['code'].max(),inplace=True)
    data['zone'].fillna(data['zone'].max(),inplace=True)
    data['extention'].fillna(data['extention'].max(),inplace=True)
    data['idx'].fillna(data['idx'].max(),inplace=True)
    data['find'].fillna(data['find'].max(),inplace=True)

    logging.info("Filling empty values with Categorical Values for coloumns (norefer,noagent,nocrawler)")
    data['norefer'].fillna(1,inplace=True)
    data['noagent'].fillna(1,inplace=True)
    data['crawler'].fillna(0,inplace=True)
    print(data.isnull().sum())
    logging.info("Missing data is handled successfully")
```

Summary metrics were then calculated for dataset including mean browser use, top searched CIKs, and number of hits per month.

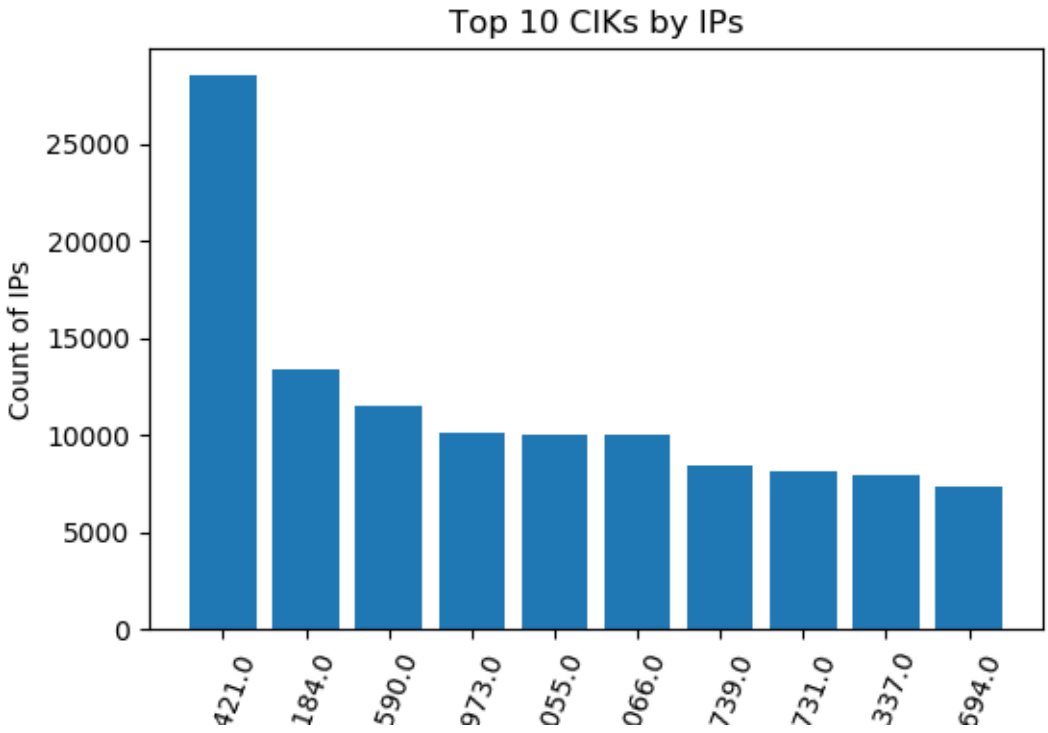
```
#SUMMARY METRICS
    logging.info("Calculating Summary metrics of clean data")
    data.describe()
    data.reset_index(drop = True)
    logging.info("Mean and Median sizes for each Browser")
    brow_df = data.groupby('browser').agg({'size':['mean', 'median'],'crawler': len})
    brow_df.columns = ['_'.join(col) for col in brow_df.columns]
    data.reset_index(drop=True)
    print(brow_df)

#To find out the 15 top searched CIKs
    cik_df = pd.DataFrame(data, columns = ['cik'])
    d = cik_df.apply(pd.value_counts)
    logging.info("Top 15 most searched CIKs with the count")
    d.head(15)
    data.reset_index(drop=True)

#Compute distinct count of ip per month i.e. per log file
    ipcount_df = data['ip'].nunique()
    logging.info("Compute distinct count of ip per month i.e. per log file")
    print(ipcount_df)

#Computing the count of status code on the basis of ip
    StCo_count=data[['code','ip']].groupby(['code'])['ip'].count().reset_index(name='count')
    logging.info("Computing the count of status code on the basis of ip")
    print(StCo_count)
    data.reset_index(drop=True)
```

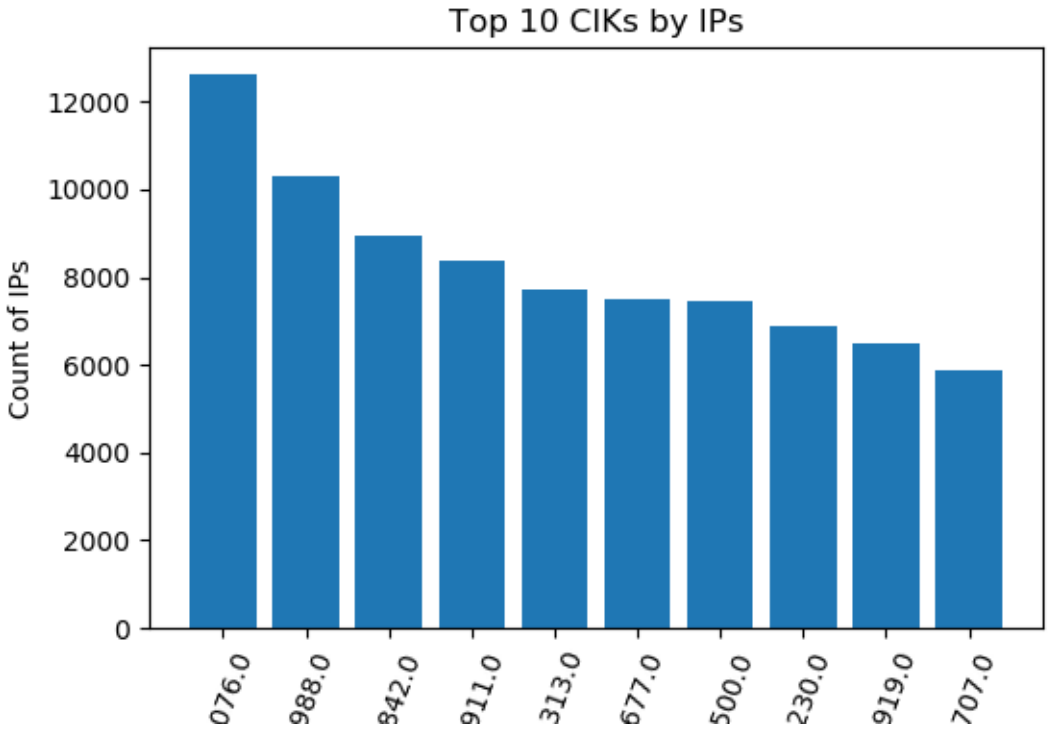
Utilizing this clean data, the application generates visualizations based on this summary data. In this case below we are creating a visualization of the top 10 CIKs by IPs searched.

```
#graph for max cik(10) by IP used
    try:
        logging.info("graphical analysis started")
        Num_of_CIKs=data[['cik','ip']].groupby(['cik'])['ip'].count().reset_index(name='count').sort_values(['count']
        data.reset_index(drop=True)
        print(Num_of_CIKs)
        u = np.array(range(len(Num_of_CIKs)))
        y = Num_of_CIKs['count']
        xticks2 = Num_of_CIKs['cik']
        plt.xticks(u, xticks2)
        plt.bar(u,y)
        plt.title('Top 10 CIKs by IPs')
        plt.ylabel('Count of IPs')
        plt.xlabel('CIK-')
        plt.savefig('Reports/CIKsbyIPcount'+ str(i) +'.png',dpi=100)
        plt.clf()
        logging.info("graphical analysis end")
    except Exception as e:
        print(u)
        logging.error(str(e))
        logging.error("Error plotting the graph ")
```

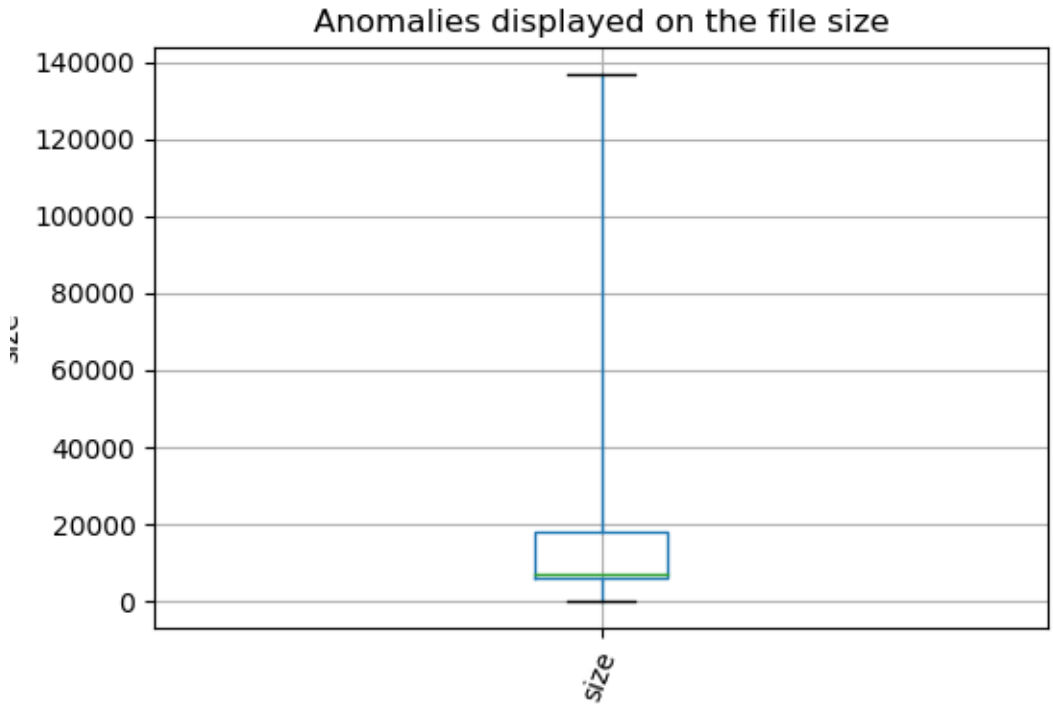
Data Analysis of 2010

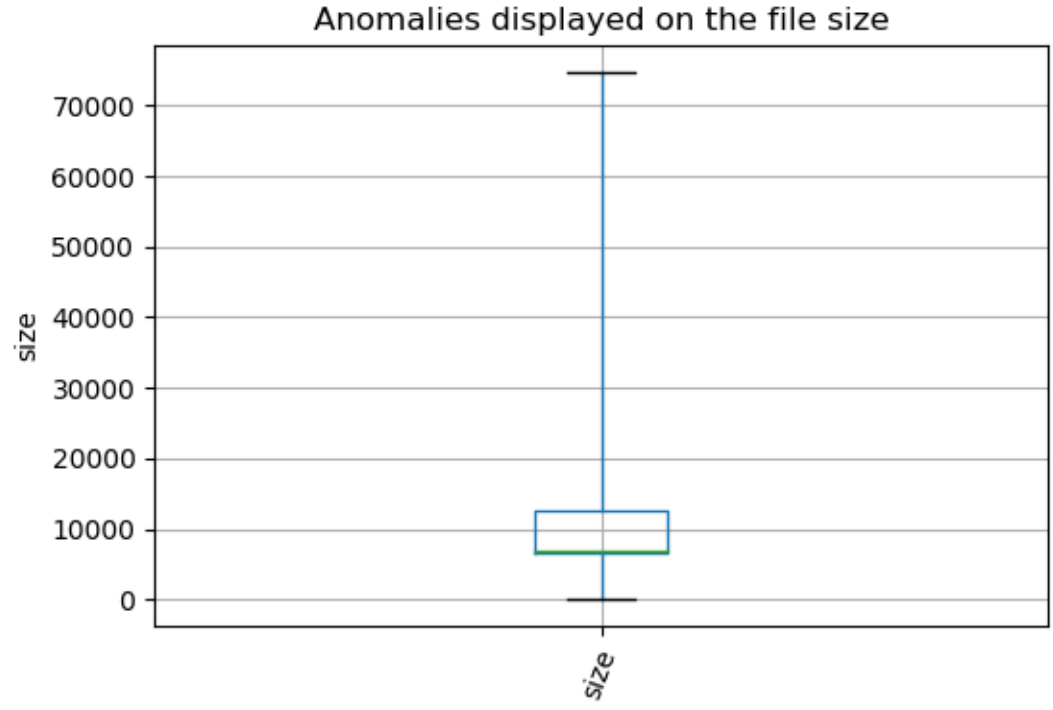
Looking at the data from 2010, we can see how the summary metrics change from month to month. Chosen summary metrics include: anomalies, CIKs by IP count, counts by browser, and file size by extension. By taking a closer look at these metrics the customer can key in on the most popular companies, which browsers are most popular, and the file size outliers from month to month.





The first graph is a graph of Nov 2010 and the bottom graph is from June 2010. When looking at the top companies searched, we can see in general is a similar amount of traffic on the top companies. The difference is that on the Nov graph, the top company is searched twice as much as the next top company.





The above graphs show file size box plots for Nov 2010 and June 2010 again. Notice how the scale changes, we can see there are larger file sizes on EDGAR in November than in June. In general the data is much more concentrated towards the bottom of

both datasets. We can see this from the median value being shown much closer to the 25% quartile.